



# Getting Started with the D0 (analysis) software

*Marco Verzocchi*

University of Maryland

part 2

## Contents

### part 1 (8:15-9:00)

- ♦ D0 software black magic explained (those setup commands)
- ♦ Software versioning and CVS
- ♦ Understanding the directory structure of different packages
- ♦ How to build an executable
- ♦ The D0 software framework
- ♦ Framework RCP

### part 2 (9:05-9:50)

- ♦ More on run control parameters (RCP)
- ♦ Understanding the RCP databases, problems with RCP files
- ♦ How to share data between packages (the EDM)
- ♦ What is the purpose of all those interfaces
- ♦ Event filtering
- ♦ Input/output
- ♦ Do and don't with the gmake command
- ♦ Using the do tools to run Do programs

## Contents

### part 3 (10:20-11:05)

- The D0ChunkAnalyze example
- Accessing some D0 physics objects from the chunks
- Writing "Elvis has just left the building" and Elvis is dead"
- Filling histograms and ROOT tuples

### part 4 (11:10-11:50)

- Other chunks
- Chunk documentation
- Trigger selection
- Stuff I wanted to cover, but didn't find time for it:
  - RTE
  - d0cuts
  - luminosity calculation and bookkeeping
  - np\_tmb\_stream
- Documentation
- Yes you can contribute.....

## A package RCP: *analysis\_example.rcp/D0ChunkAnalyzer.rcp*

```
// Author: Marco Verzocchi
// Date: 7/24/02
//
// $Id: D0ChunkAnalyze.rcp,v 1.2 2002/08/16 16:22:25 mverzocc Exp $
//
// String PackageName = "D0ChunkAnalyzer"
//
```

```
namespace D0example {
    class D0chunkAnalyze : public fwk::Package, public fwk::Process,
                           public fwk::Analyze, public fwk::JobSummary,
                           public fwk::FileOpen, public fwk::FileClose {
public :
    // Constructor/Destructor
    D0chunkAnalyze(fwk::Context* context);
    ~D0chunkAnalyze();

    // Overridden hook methods
    fwk::Result processEvent(edm::Event &event);
    fwk::Result analyzeEvent(const edm::Event &event);
    fwk::Result fileOpen(const fwk::FileInfo &fileinfo);
    fwk::Result fileClose(const fwk::FileInfo &fileinfo);
    fwk::Result jobSummary();

    // Method for checking whether a trigger has fired.
    bool TriggerFired(const edm::Event &event);
}
```

The framework  
knows that when  
it finds a given  
PackageName in a

framework RCP file  
it has to create an

instance of the class  
*.../analysis\_example/D0ChunkAnalyzer.hpp*

*D0 (analysis) software tutorial*

10/8/02

4

# What happens if you ask for something and it's not available ?

In *analysis\_example/rcp/runD0ChunkAnalyze.rcp* there are two lines:

```
string Packages = "geometry read config unpack trigsel analyze"
RCP trigsel = <analysis_utilities D0TriggerFilter>
```

and in *analysis\_example/bin/OBJECTS* there is a line *RegD0TriggerFilter*

I removed that line, recompiled *DoChunkAnalyze\_x* and tried to run again....



The framework tries to build an instance of the *D0TriggerFilter* object, I am requesting it .....

```
%@%@%@%@%ERLOG-A init: Framework Unable to make package of type
D0TriggerFilter
    with name trigsel
        The name comes directly from the framework RCP file.
        / 6-0ct-2002 19:56:32 Controller:/ Beginning of job
%ERLOG-A init: Framework constructor: exception
Framework 6-0ct-2002 19:56:32 -:- Beginning of job
DoChunkAnalyze_x.log lines 1-6/6 (END)
```

A pretty common error

## A package RCP: *analysis\_example\_rcp/D0ChunkAnalyzer.rcp*

```
// Author: Marco Verzocchi  
// Date: 7/24/02  
//  
// $Id: D0ChunkAnalyzer.rcp,v 1.2 2002/08/16 16:22:25 mverzocci Exp $  
  
string PackageName = "D0ChunkAnalyzer"  
  
// Use this flag to print debug information.  
untracked bool debug = false
```

→ Untracked variable

The **RCP** files can be used to **control the behaviour of packages**, passing some input parameters (for example radius of the cone used for finding jets ....). Variables (**int**, **float**, **bool**, **string** and **arrays of ....**) may be **untracked** (the RCP database doesn't store them) or tracked (**stored in the RCP database and eventually in the event itself**)

```
// Create and fill histograms and file used for saving them.  
bool FillHistograms = true  
string HistogramFile = "D0ChunkAnalyzer.root"  
  
// Event tag given to selected events. Use this tag for writing  
// out only selected events in the DST/TMB/R00Ttree formats  
// (using the WriteEvent, Thumbnail and TMBTree packages).  
string Event_tag = "D0ChunkAnalyzer.tag"  
D0ChunkAnalyzer.rcp Lines 1-31/31 (END)
```

## RCP databases

**RCP parameters are stored in databases** (which can be flat ASCII files on disk). Each RCP file is identified by two numbers (ex RCPID <1 99>, this is something in a private area, or RCPID <3721 2>, this is something in the build area, or RCPID <13574 1>, something in the production area).

**Who cares ?** You ..... **Why ?** While reviewing your paper draft, Tom Ferbel claims that there must be something wrong with your jet finder, you think that you've done the right thing, but you've lost all your log files.... **How do you figure out how you reconstructed your jets ?**

The **RCP identifier** used by a certain package to create jets is **stored** along with the jets themselves. As long as your database is still there, you can figure out how the jets were reconstructed

## A package RCP: *analysis\_example.rcp/D0ChunkAnalyzer.rcp*

These 3 lines in the RCP file will actually be used to tell the code that we want the jets found by the Run II cone algorithm run with a 0.5 DR radius and preclustering (a.k.a. JCCB)

```
// Select the jet finding algorithm  
// (Run II 0.5 cone algorithm with preclustering)  
// See the RCP files in the jetanalyze package for other possibilities.  
string JetAlgo_type = "PreSCilcone"  
string JetAlgo_names = ("towers" "coneSize" "Radius_of_Cone" "Min_Jet_ET")  
float JetAlgo_values = ( 0.3 0.5 8. )
```

```
// Select the EMparticle chunk:  
// simple cone algorithm --> EMReco-scone-id  
// cell nearest neighbour algorithm --> EMReco-cnn-id  
RCP EMid_Algo = < emreco EMReco-scone-id >  
string EMid_SearchRCPs = ("clusterer", "EMReco")
```

This solves the problem of keeping track of who has done what....

These 2 lines in the RCP file are used to tell the framework that we want the EM objects found with the simple cone algorithm

**Other things which can go wrong with RCPs .....**

If everything is in a public database, you should be able to recover the RCP identifiers, and go back to what has been done

**What is in the public databases ?**

## FROZEN production and development releases

FROZEN is the key word..... you SHOULD NOT trust the **content of the RCP database until the release is frozen** you SHOULD NOT trust the content of your private DB (the RCP you modify in your working area)

RCPID < 1 99>

if you've worked one day with a **non-FROZEN db** and the next day things don't work, this may help.....

```
$> rm -r rcpdb  
$> dosetup
```

2

Attempt to add a parameter set known by RCID: 1185 2 to a parameter set under construction.

This RCID is not known to any of the RCP databases in use  
This problem encountered while reading the script or DB file:

**How do different packages share data ?**  
**How do programs share data ?**

**EDM  
DoOM**

**What is an "event" ? Area in memory (the pellets) in which you can store information.** Each pellet (called chunk) has different barcodes on it (identifiers):

- **chunkID** (type: jets, tracks, .....)  
(how you created this specific jet)
- **rcpID** (this may depend on the environment)
- **envID** (run and event number !!!)
- **collisionID**

Each chunk is accessed in a standard way (the forklift) and can be identified (with a barcode reader)....

Chunks can talk to each other (pointers from tracks to electrons, from electrons to calorimeter cells,.....)

**Features of EDM are best discussed through examples (part 3)**

**D0OM** is the package which is used to store data on disk  
It knows how to read and write a class, how to deal with  
pointers (addresses in memory may change). It also allows you  
to pack informations....

An aside:

**DST** (data summary tapes) contain the output of the  
reconstruction program (D0reco). **Minimal data compression,**  
**no information loss (150 KB/event)**. Can be read directly into  
memory (chunks).

thumbnails (TMB) contain packed data (**compression**) with  
**some loss of information** (do not store hits on tracks). **Small**  
**size** (<10 KB per event, can keep all data on disk). **Must be**  
**unpacked and chunks must be recreated** (some quantities have  
to be recalculated).

We've mentioned interfaces twice so far.....

## → List of interfaces defined and actually used

```
string Interfaces = "generator decide builder filter modify analyze process dump  
jobSummary runEnd runInit fileOpen fileClose"  
string Flow = "generator decide builder filter modify process analyze dump"
```

```
namespace D0example {
```

```
    class D0ChunkAnalyze : public fwk::Package, public fwk::Process,  
        public fwk::JobSummary,  
        public fwk::FileOpen, public fwk::FileClose {
```

Interfaces implemented in the  
**D0ChunkAnalyze** package

```
// Overridden hook methods  
fwk::Result processEvent(edm::Event &event);  
fwk::Result analyzeEvent(const edm::Event &event);  
fwk::Result fileOpen(const fwk::FileInfo &fileInfo);  
fwk::Result fileClose(const fwk::FileInfo &fileInfo);  
fwk::Result jobSummary();
```

```
// Method for checking whether a trigger has fired.  
bool TriggerFired(const edm::Event &event);
```

```
// Overridden package methods  
std::string packageName() const {return package_name();}  
static const std::string package_name() {return "D0ChunkAnalyze";}  
.../analysis_example/D0ChunkAnalyze.hpp lines 43-67/144 45%
```

## What is the purpose of all those interfaces ? (1)

- constructor
    - initialize your package, read the package RCP file(s), create histograms and tuples (not called by the framework, don't put anything important here)
  - destructor
  - jobSummary
    - print summaries of processing, calculate efficiencies, save histograms
  - fileBegin
    - just tells you a new file was opened, you can do your own bookkeeping
    - the last event in a file has been processed, if you want you can do file summaries encountered a new run, used to load calibration constants from database(s)
  - fileEnd
  - runInit
  - runEnd
    - last event in a run, make run summaries
- NB Given the was SAM works, you may be delivered files from run A, then from run B, then again from run A**

## What is the purpose of all those interfaces? (2)

**REMEMBER:** not all interfaces need to be implemented !

interfaces which may modify the event (add chunks)

- buildEvent      read an event from the input file or generate it using the MC
- processEvent      read some chunks, do something, write other chunks (**this is where most of the processing goes**)
  - more processing
- last chance of adding something to the event
- modifyEvent
- finishEvent

**Most packages put almost all their processing in processEvent**

If everything went fine all interfaces should finish with

```
return Result:success;
```

**Any other result will halt processing for the current job.**

## What is the purpose of all those interfaces ? (3)

interfaces which may not modify the event (event is const)

- filterEvent
- put here a user decision on whether to continue processing for this event or not (*return Result::failure;* in this case, exception to the rule stated on the previous slide)
- analyzeEvent
- put here your analysis code (histogram filling)
- dumpEvent
- code to dump the chunk(s) created by this package  
rarely used
- tagEvent
- outputEvent
- makeDecision
- write out the event  
rarely used

## Which interfaces shall I use ?

- constructor
  - filterEvent
  - processEvent
  - analyzeEvent
  - jobSummary
- initialize your package  
possibility of doing event filtering  
processing and modifying the event  
processing but no modification to the event  
a summary at the end is always nice

## Example:

- **select events** with 1 high  $p_T \mu$ , 4 jets, 2  $b$ -tag, and  $\cancel{E}_T$ , throw away the remaining events
- **reconstruct** the two top quarks and add them **to the chunk**
- **processEvent**
- **analyzeEvent**
- **jobSummary**

## I/O in D0 framework executables:

- 1) need a few entries in ...../**bin/OBJECTS** (part 1 slide 35)

input (raw data, DST, thumbnails):

- 2) **input from disk:** need **one** entry in the framework RCP file

**RCP read = <d0reco D0recoReadEvent>**

- 3) **input from SAM:** need **two** entries in the framework RCP file

**RCP sam = <d0reco D0recoSAMManager>**

**RCP read = <d0reco D0recoReadEvent>**

**no need to change the D0recoReadEvent.rcp file, use d0tools**  
for specifying the list of input files (case 1) or the SAM dataset  
definition (case 2)

## Output (raw data, DST, thumbnails):

- 4) **write out a DST file (input from DST)**  
*RCP dstwrite = <analysis\_example WriteEventDST>*
- 5) **write out a TMB file (if reading a DST, create the thumbnail chunk in memory first !!!)**  
*RCP tmbwrite = <analysis\_example WriteEventTMB>*

**very few differences (name of output file(s), number of events per file) but look at the second part of the RCP file:**

```
// Output filter stuff
// Use parameter VetoClassList to drop a specified set of classes or chunks.
// Use parameter WriteClassList to write only a specified set of classes
// (class edm::Event and edm::EventValue are always retained). Use parameter
// WriteChunkList to write only the specified set of chunks. Parameter
// WriteClassList is not generally very useful, because it is necessary to
// list every class that is desired to be written, not all of which may
// be obvious. However, parameter WriteChunkList provides a practical way
// to eliminate all but one, or a few, chunks from an event.

bool UseOutputFilter = true
string VetoClassList = ""           // List of chunks not written on output. Use
                                   // an empty list to write all chunks.

string WriteChunkList = "MCKineChunk TMBTriggerChunk thumbnail::ThumbNailChunk"
                                   // Write only the specified chunks. Use an
                                   // empty list to write all chunks.

string WriteClassList = ""           // Write only the specified classes, plus
                                   // edm::Event and edm::EventValue. Use an
                                   // empty list to write all classes.
```

`WriteEventTMB.rcp` lines 17-38/58 70%

## Even more important: write out only selected events

```
// Event tagging.  
  
string WriteEventTags = ""  
    // Only write events tagged by at least one  
    // of the specified tags. Use "*" or an empty  
    // list to select all events for writing.  
    // Exclude events tagged by any of the  
    // specified tags.  
  
// Chunk tagging.  
  
string WriteChunkTags = ""  
    // Only write chunks tagged by at least one  
    // of the specified tags. Use "*" or an empty  
    // list to select all chunks for writing.  
    // Exclude chunks tagged by any of the  
    // specified tags.  
  
// Write out unknown chunks  
  
bool CopyUnknown = true  
WriteEventIMB.rcp Lines 39-58/58 (END)
```

In the code tag one nice event (passing your cuts) with the following line:

`event.tag("myHiggsCandidate");`

in the RCP file put

`string WriteEventTags = "myHiggsCandidate";`

only those events will be written to the output stream

## More on output files:

- can have multiple output streams (one per event tag type ?),  
just need **multiple instances of the write package**  
(**write1**: electron candidates,**write2**: muon candidates,...)

## ROOT output

- from **RecoAnalyze** (ROOT tuples, discontinued in p13.00.00)
- from **TMBAnalyze** (ROOT tuples and **ROOT trees**)
  - need all the objects which are in *tmb\_analyze/bin/OBJECTS*
  - need the RCP parameters which are in  
*tmb\_analyze/rcp/runTMBTreeMaker.rcp*
- only **ONE output stream**
  - no event tagging

## More on event tagging

- A tag can be added into the event at any time during the processing (filterEvent, processEvent, analyzeEvent,...)  
• And it can be queried too:

```
if event.hasTag("myHiggsCandidate") {  
    E-mail.send("Klima@fnal.gov");  
}
```

- It can be used for selective processing:

- If I find a  $J/\psi$  I may try to see whether I can find a  $K^+$  coming from the same vertex that reconstructs a  $B^+$ 
  - I may need to lower the  $p_T$  threshold on the tracking for this particular event

## Using gmake (good things to do and bad things to avoid)

### Compile packages and create executables

```
$> gmake all
```

```
$> gmake analysis_example.all
```

### Test packages

```
$> gmake test
```

```
$> gmake analysis_example.test
```

### Clean up

```
$> gmake clean
```

```
$> gmake analysis_example.clean
```

### Things to avoid (particularly after a cleanup)

```
$> gmake analysis_example.lib
```

## Using d0tools for running D0 programs

Figuring out all the options for some programs (D0reco, RecoAnalyze, D0analyze, D0TrigSim,...) may be complicated:

- **d0data/MC**
- input from disk/SAM
- single/multiple files
- with/without debugger
- **interactive/batch**
- **d0mino/clued0/CAB**

Nice script to sort out a lot of these issues:

```
$> setup d0tools  
$> rund0exe -h
```

This brings up an exhaustive help page, documenting a lot of options ..... *rund0exe* is a backend for other scripts: *runreco*, *runrecoanalyze*, *rund0analyze*, *runD0TrigSim*, *runScriptRunner*, *runchunkanalyze* (since they share a lot of the options)

Command to run D0 framework executable.

Usage: rund0exe [...]

Command line options:

General

-h – Get help.

Global parameters

-exe=exe	– Name of executable
-rcp=rkp	– Name of framework rcp script
-rcppkg=rcppkg	– Name of framework rcp package
-initscript=script	– Initialization script for executable
-userscript=script	– User provided initialization script for executable
-localbuild	– User provided initialization script for executable
-localrcp	– Use local build instead of official version
-localfwrkp	– Use local rcp's in addition to official ones
-name=name	– Use local version of main framework rcp
	– Override default job name

Lines 1-26

Specify the name of the program you want to run, the framework RCP file and the name of the package in which the framework RCP file is located

Some programs need additional input data files, use the scripts to copy them into the working area

Use local (built/modified by the user) version of the program or of the RCP files, instead of the one taken from the official software repository

# More options, batch, debugging, optimized/non optimized code

Submit a batch job

Modes of running (default mode is interactive)

- batch - Use batch queue [default = short]
- debug - Use interactive debugger.
- purify - Use Purify.
- profile - Use srun to profile (works on domino only).
- maxopt - Use optimized build.
- jobname=jobname - Specify batch job name [default = rund0exe]

Start the debugger  
(Totalview)

Use maxopt version

- domino (LSF) batch system switches:
- q=queue - Specify default batch queue.
  - mem=mem - Specify maximum memory. [default = 500000]

Above switches ignored if SAM is being used (SAM picks the queue)

clued0 (PBS) batch system switches:

- cputime - Specify maximum CPU time. [default = 3:00:00]
- mem=mem - Specify maximum memory. [default = 500Mb]
- node=node - Specify node where job should be run [default = system chooses]
- thisnode - Run job on the current node.

## Input/output: list of files or SAM dataset definition

Options controlling input / output

-filelist=filelist.dat - List of files/datasets to process

-defname=defname - SAM project definition name

-out=file - Write processed events to specified file

Options to control which events are processed

-num=num - Specify number of events to process [default = all]  
-skip=num - Specify number of events to skip [default = none]  
-eventlist=eventlist.dat - Process only these specific collision ids  
eventlist.dat is a file containing a list  
of <run> <event> collision ids.  
-skipruns=baddruns.dat - Skip these runs  
baddruns.dat is a file containing a list of run numbers  
-skipevents=badevents.dat - Skip these specific collision ids  
badevents.dat is a file containing a list  
of <run> <event> collision ids.

Lines 63-85

Options controlling the number of events to be processed,  
the events/runs to be skipped

### Special options

-fpe - Turn on floating point exception handling on Linux

-runscript=script - Specify a custom script to run

The specified script will be run before actually running the executable. Note that this script will only work reliably if it contains command line commands. It is intended for advanced users.

-fwkparams - Pass any set of parameters to a framework executable

You may pass any set of parameters to the framework executable you are running by using the -fwkparams option. This option should be the last one on the command line, followed by the actual parameters you want to pass to the framework. For example.

```
rund0e... -fwkparams -option1 xxx -option2 yyy
```

lines 105-123

Used to invoke scripts which perform actions needed prior to running a program (copying data files, generating trigger lists, ....)

→ Can be used to pass additional options to programs (if the code knows how do parse the command line): used for example to pass a list of triggers used for event filtering in the analysis\_example code (slide 33)

**Many other options, read the documentation (it's well written)**

Normally use simpler scripts: *runreco*, *run.....*

**Some of these scripts have special options:**

- if the **input is from SAM** you must use a framework RCP which starts **SAMManager** (all the *runxxx* scripts)
  - if your code needs to do **different things for data and MC** you must tell it somehow (*-format=data/MC* option in many *runxxx* scripts)
  - if you're **reading from a thumbnail** you have to unpack the **data** (not the case if you're reading a DST, *-mode=DST/TMB* option in *runchunkanalyze*)
  - You may want to run just L1/L2 or L3 trigger simulations (and you can rerun the trigger on data offline, several additional options in *runD0analyze*, *runD0TrigSim*, *runScriptRunner*)
- Read the man pages:** *setup d0tools; runxxx -h*

## *runchunkanalyze* specifics:

**you MUST compile this code, it is just an example, you are supposed to do some homework and try to add your own histogram to it, so you also have to compile it otherwise it will complain**

```
[36] mverzocc@cole-clued0:/home/mverzocc>runchunkanalyze -h
```

Command to run the D0ChunkAnalyze example job.

Usage: runchunkanalyze [....]

Example

First of all you have to build the D0ChunkAnalyze executable (and customize it, this is supposed to be your analysis job).

For example, using the p12.02.00 production release, you should:

```
% setup n32          # on domino
% setup D0RunII p12.02.00
% setup d0cvs
% cd <working_area>
% newrel -t p12.02.00 EXAMPLE
% cd EXAMPLE
% addpkg analysis_example
% d0setwa
% srt_setup SRT_QUAL=maxopt # turn on optimization
% gmake all
--More--
```

**You've already seen/  
done this**

To process a thumbnail file (TMB):

```
% runchunkanalyze -filelist=mydata.dat -mode=TMB [-maxopt]
```

To process a DST file:

```
% runchunkanalyze -filelist=mydata.dat -mode=DST [-maxopt]
```

To process thumbnail files in SAM:

```
% runchunkanalyze -defname=my-dataset-definition-name \
-mode=TMB [-maxopt]
```

To process DST files in SAM:

```
% runchunkanalyze -defname=my-dataset-definition-name \
-mode=DST [-maxopt]
```

--More--

This is easy: read either a DST or a thumbnail (TMB)  
either from disk or SAM

and yes, always use the maxopt option

## Fancier options: skim the data (i.e. read all events in input, apply some cuts, write the events out): uses the event tag mechanism (earlier in these slides)

- skim=output\_type
  - Produce a skinned set of events in output.

This option instructs the program to write a skinned dataset in one (or more) formats, using an event selection based on triggers and eventually user cuts.

A custom RCP file is created on the fly, according to the user requirements. The following type of datasets are available in output:

DST - write a skinned DST (only if using a DST as input)  
TMB - write a skinned TMB  
ROOT - write a skinned ROOT tree

To create multiple output files use either the -skim=type1,type2 or the -skim=type1 -skim=type2 syntax.

To perform a selection based on physics quantities, insert in the code of D0ChunkAnalyze (analysis\_example/src/D0ChunkAnalyze.cpp) your selection code and add a tag to the event for the selected events, using the following code:

```
event.tag("mytag");  
Then in the following RCP file:  
analysis_example/RCP/WriteEventTMB.rcp  
add the tag "mytag" to the line:  
string WriteEventTags = "...";  
(if you want to skim DST data, change the WriteEventDST.rcp file).
```

-More--

NB. THIS OPTION IS CURRENTLY NOT AVAILABLE FOR THE ROOT TREE

Not for the ROOT tree

More options: I don't know what to put in the RCP for SAM,etc, but I want to customize the RCP file: run `runchunkanalyze` twice, the first time with the `-create_rcp=....` option, this will give you a **custom RCP file which you can edit further**, then run using the custom RCP

`-create_rcp=file.rcp` – Create a custom RCP file, without running D0ChunkAnalyze

Uses all the `-mode`, `-filter_triggers` and `-skim` options to create a custom RCP file in the `analysis_example/rcp/` directory, with the user provided name. D0ChunkAnalyze is not executed, the user is given the opportunity to customize further the RCP file. D0ChunkAnalyze can be run later using the `-rcp` option.

`-rcp=file.rcp` – Run D0ChunkAnalyze with a custom RCP framework file

Ignore the `-mode`, `-filter_triggers` and `-skim` options and run D0ChunkAnalyze with a custom RCP file from the `analysis_example/rcp` directory, with the user provided name. This file has been built by running first with the `-create_rcp` option, and then customized further by the user.

**=More=**

Try it, if you want to see what goes in the framework RCP for the other options...

## Last feature: filtering or tagging events based on trigger masks

**-filter\_triggers=EM\_HI\_EM\_HI\_EM5**: skip the processing for events which do not pass either of these triggers (event filtering)

-filter\_triggers=triglist – Comma separated list of triggers used for filtering events.

Specify a list of L3 trigger names (separated by commas) used to filter events during the processing (requires that the executable is linked with RegTriggerFilter)

-tag\_triggers=triglist – Comma separated list of triggers used for event tagging.

Specify a list of L3 trigger names (separated by commas) used to insert an EDM tag in events during the processing (requires that the executable is linked with RegTriggerFilter)

**-tag\_triggers=EM\_HI\_EM\_HI\_EM5**: tag the events which pass one of these two triggers (event tagging). You may do something special for these events (write them to a separate stream....)

These options are available to any D0 program, provided you add RegDOTTriggerFilter in the OBJECT file, you can use these options with *-fwkparams -filter\_triggers/-tag\_triggers=....* in *rund0exe*

## How do I create $\pi\epsilon\tau\rho\sigma$ \_ $\alpha\gamma\lambda\zeta\epsilon$ ?

```
| Documentation is available at: http://www-clued0.fnal.gov/clued0
| Do software release installed: type 'd0rel' to see which releases
| firewall activated: only www-clued0 will be visible from outside FNAL
| DO 09/10/01
|
[31] mverzocc@cole-clued0:/home/mverzocc>cd EXAMPLE
/lwork/cole-clued0/mverzocc/EXAMPLE
[32] mverzocc@cole-clued0:/w0rk/cole-clued0/mverzocc/EXAMPLE>setup D0RunII p12.03.00
se*** setup D0RunII p12.03.00
tup d[33] mverzocc@cole-clued0:/w0rk/cole-clued0/mverzocc/EXAMPLE>ctnewpkg -1 petros_analyze
[34] mverzocc@cole-clued0:/w0rk/cole-clued0/mverzocc/EXAMPLE>ctnewpkg -1 petros_analyze
Creating package petros_analyze
Creating VERSION (00.00.00)
Creating directory src
Adding SUBDIRS
Adding src to COMPONENTS
Creating file src/COMPONENTS
Creating directory doc
Creating documentation file doc/index.html
Please edit doc/index.html appropriately
[35] mverzocc@cole-clued0:/w0rk/cole-clued0/mverzocc/EXAMPLE>cd include
[36] mverzocc@cole-clued0:/w0rk/cole-clued0/mverzocc/EXAMPLE/include>ln -sf ../petros_analyze/petros_
analyze petros_analyze
[37] mverzocc@cole-clued0:/w0rk/cole-clued0/mverzocc/EXAMPLE/include>
```

You need to do two things:

```
$> ctnewpkg -l  $\pi\epsilon\tau\rho\sigma$  _  $\alpha\gamma\lambda\zeta\epsilon$ 
$> cd include; ln -sf ../ $\pi\epsilon\tau\rho\sigma$  _  $\alpha\gamma\lambda\zeta\epsilon$ / $\pi\epsilon\tau\rho\sigma$  _  $\alpha\gamma\lambda\zeta\epsilon$ 
```

Now you have an empty package with the right structure, it's up to you to fill it correctly

**End of part 2  
reconvene at 10:20  
(sharp)**